C program calculates checksums

Ken Levine, Airshow Pacific Systems, Kirkland, WA

To ensure data integrity, it's wise to frequently calculate file checksums. The C program in **Listing 1** calculates the checksum of a file using a 16-bit CRC (cyclic redundancy check). The program assumes an 8-bit byte size. The routine reads the file as binary and processes one byte at a time. The CRC formula the program uses is X16+X12+X5+1, with a starting value of hexadecimal FFFF. The program displays the number of bytes processed. The file calccrc.c starts by including the header files needed. Next, the routine defines and initializes the constants. (The program does not use the C++ keyword const; therefore, a C compiler can compile the program.) Inside main(), the program defines variables and issues the initial starting value for the CRC. The routine performs a check to verify that at least two arguments passed to main(). If only one argument passes, the program terminates with a message that you need to supply a file name.

The program reads and processes one byte at a time until it reaches the end of the file. Each time the program reads a byte, the byte counter increments. At the end of the file, the program displays the CRC of the file and the number of bytes read. The routine calculates the 16-bit CRC, byte by byte, using the calcCRC16 function. This function passes the byte to be processed and the current value of the CRC and returns the new CRC value. The program calculates the CRC for each bit in the byte. The variable temp is assigned the current CRC value right-shifted 15 times, XORed with the current byte value right-shifted seven times. This operation XORs the MSB of the CRC with the MSB of the byte, so temp will have the value zero or one. Note that this operation does not change the values of the CRC or the byte. The CRC then left-shifts one place. If temp is 0, nothing happens. If temp is 1, the program XORs the CRC with hexadecimal 1021 (the X12+X5+1

LISTING 1–C PROGRAM FOR CALCULATING CRC

```
/* file calccrc.c
 * calculates the 16 bit CRC of a file
 * in Borland C++, an int is 16 bit
 *
 * file should be read in binary as this will read every character,
 * when a file is read in binary mode, the carriage return (CR) and
 * line feed (LF) are both read; in text mode, only the LF is read
 * should a file have CR in part of the data, text mode may not read
 * it
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <fcntl.h>

unsigned int INIT = 0xFFFF;  /* initial value of CRC */
unsigned int CRC16 = 0x1021; /* bits 12, 5 and 0 */
int SHIFT_CRC = 15;          /* how far to right shift crc */
int SHIFT_BYTE = 7;          /* how far to right shift byte */
int BYTE_SIZE = 8;           /* number of bits in a byte */


unsigned int calcCRC16(unsigned int, unsigned char);

int main(int argc, char *argv[]){

      unsigned int crc = INIT;
      long byteCounter = 0;
      FILE *Data;
      char ch;

if(argc  < 2){
            printf("\nNeed to supply file name for CRC calculation.");
            printf("\nProgram terminated.");
            exit(1);
      }

/* see if file can be opened*/
      if(NULL == (Data = fopen(argv[1], "rb"))){
            printf("\nUnable to open file %s, program terminated.", argv[1]);
            exit(1);
      }

      while(!feof(Data)){
            /* The end of file character is read and processed
             * by these statements.
            */
            ch = fgetc(Data);
            crc = calcCRC16(crc, ch);
            byteCounter++;
      }

      fclose(Data);
      /* Subtract one from byteCounter to compensate for the
       * end of file character being read. This character is

             * not counted when the operating system shows file
             * size.
            */
         byteCounter--;

            printf("\ncrc of %s is %x",argv[1], crc);
            printf("\n(read %ld bytes)", byteCounter);

            return 0;
      }


      unsigned int calcCRC16(unsigned int crc, unsigned char byte){

      /* Algorithim XOR's bit 15 (the MSB) of the current CRC with the current
       * MSB of byte.  The current CRC and byte are then left shifted by one.
       * This value is then XOR'ed with bits 12, 5 and 0 of the current CRC.
       * This is done until all bits in byte have been processed.
       */
```

(continued on pg 152)

term). Next, the byte left-shifts one place, so the program can process the next bit. This process repeats until the routine processes all eight bits of the byte.

If you wish to use a different formula to calculate the CRC, you need only change the variable CRC16, assuming that the formula still starts with the X16 term). If you wish to calculate a 32-bit CRC, the variables INIT and CRC16 cannot be of type int. You can set the variables SHIFT_CRC, SHIFT_BYTE, and BYTE_SIZE to other values to accommodate various byte and CRC sizes. This program is compiled using Borland C++ 3.0, Borland C++4.5, and Microsoft Visual C++ 1.0. You can download **Listing 1** from *EDN*'s Web site, www.ednmag.com. Click on "Search Databases" and then enter the Software Center to download the file for Design Idea #2674.

LISTING 1–C PROGRAM FOR CALCULATING CRC *(continued)*

```
        unsigned int temp;
        int index;

        for(index = 0; index < BYTE_SIZE; index++){
              temp = (crc >> SHIFT_CRC) ^ (byte >> SHIFT_BYTE);  /*temp is now MSB
of CRC X or'd
         with MSB of byte */
              crc <<= 1;  /* left shift one space */

              if(temp){  /* if temp is 1, then XOR bits 12, 5 and 0 with 1
                                    * if temp is 0, no need to XOR because XOR
with 0
                                    * does not change value */
                    crc ^= CRC16;
              }

              byte <<= 1; /* left shift one to get to next bit */
        }

        return crc;
}
```

Is this the best Design Idea in this issue? Vote at www.ednmag.com/ednmag/vote.asp.

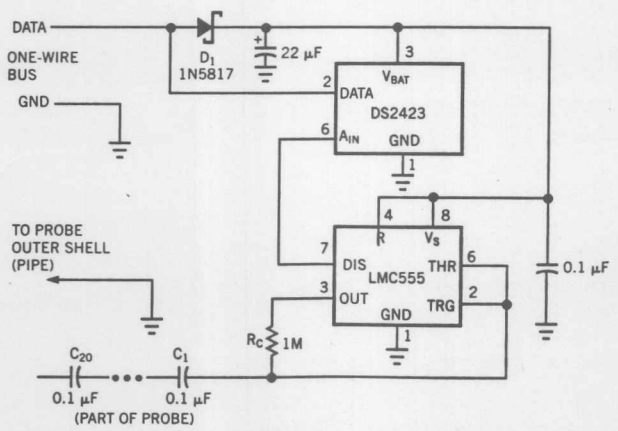
One-wire bus powers water-level sensor

Dale Litwhiler, Lockheed Martin, Newtown, PA

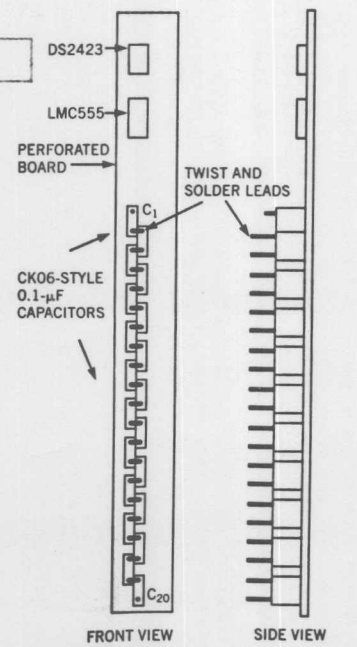
You can use the simple sensor circuit in **Figure 1** to remotely monitor the level of liquid water in a vessel such as a swimming pool. The LMC555 sensor oscillator provides an output-signal frequency that is a function of the water level. This signal drives a DS2423 pulse counter. A host PC or µC reads the output of the pulse counter via the Dallas Semiconductor one-wire bus (**Reference 1**). The circuit uses approximately 150 µA of current, allowing the circuit to steal its power from the bus via the Schottky diode, D_1. Because the circuit is sensing water that is part of the electrical circuit, you should use an ac-coupled signal to avoid polarization of the water and plating of the electrodes. One approach is to have the water in a circuit branch that is in series with some capacitance. In this sensor circuit, the water is in the branch containing the timing capacitance of a CMOS 555 timer, configured as a free-running oscillator with a 50% duty cycle. The sensor provides a capacitance that varies with water level.

Figure 2 shows one method of fabri-

Figure 1



A simple 555-type timer and a counter form the heart of a water-level sensor.



A string of series-connected capacitors provides an indication of water level.